Wrocław, 28 kwietnia 2024

Paradygmaty Programowania Obiektowego

Laboratorium 4

Polimorfizm, metody wirtualne, klasy abstrakcyjne, interfejs

prowadzący: *mgr inż. Marta Lampasiak*

1 Wprowadzenie

Celem laboratorium jest poznanie, omówienie oraz przećwiczenie zagadnień i mechanizmów takich jak: polimorfizm, metody wirtualne, klasy abstrakcyjne, interfejs.

Polimorfizm jest jednym z paradygmatów programowania obiektowego. Jest on powiązany z tzw. **metodami wirtualnymi**. Cecha ta umożliwia różne zachowanie tych samych metod wirtualnych w czasie wykonywania programu.

Podczas dziedziczenia obiekt klasy pochodnej może być wskazywany przez wskaźnik typu klasy bazowej. Natomiast typem statycznym obiektu wskazywanego przez wskaźnik jest typ tego wskaźnika. Typem dynamicznym obiektu wskazywanego przez wskaźnik jest typ na jaki dany wskaźnik wskazuje.

Metoda wirtualna to specjalna składowa klasy, która przydaje się szczególnie, gdy używamy obiektów, które posługują się wskaźnikami lub referencjami do niego. W przypadku użycia zwykłej metody, która podlega przesłanianiu w innych klasach to, jaka metoda zostanie wywołana (z klasy bazowej, czy pochodnej) zależy od typu wskaźnika (typ statyczny). Jeśli chcemy, aby wywołanie metody zależało od tego na co wskazuje wskaźnik (typ dynamiczny) musimy użyć metody wirtualnej. Jeżeli chcemy, aby jakaś metoda była wirtualna to należy przy jej deklaracji w klasie bazowej dopisać słowo *virtual*.

Jedną z zasad, którą warto zapamiętać jest to, że jeżeli używamy chociaż jednej metody wirtualnej, to **destruktor musi być również wirtualny**. Niewirtualny destruktor może powodować wycieki pamięci i niezdefiniowane zachowanie programu, ponieważ będzie wywoływany ten, który odpowiada typowi wskaźnika, a nie ten, który odpowiada rzeczywistemu typowi obiektu.

```
1 class MojaKlasa
2 {
3 public:
4     virtual ~MojaKlasa(); //wirtualny destruktor
5 };
```

Generalnie z punktu widzenia samego kompilatora wystarczy dodanie słowa *virtual* przy deklaracjach metod tylko w klasie bazowej. Natomiast można je również dodać w klasie pochodnej lub w miejscu wystąpienia definicji wirtualnej funkcji składowej, jeżeli taki kod jest dla programisty czytelniejszy.

Używając metod wirtualnych należy liczyć się z tym, że powodują one utratę wydajności oraz narzut pamięciowy. Kiedy mamy typ statyczny, to odpowiednia metoda jest już znana i wybierana na poziomie kompilacji, ponieważ kompilator wie, która metoda ma zostać użyta. Natomiast kiedy mamy typ dynamiczny, to kompilator nie jest w stanie na poziomie kompilacji określić, która z metod ma zostać użyta. Zostanie to określone dopiero na etapie działania programu.

Każde wywołanie metody wirtualnej wiąże się z dodatkowym narzutem czasowym w trakcie wykonania. Wybieranie właściwej metody to nie tylko czas, ale również potrzebny jest dostęp do informacji o różnych wersjach metody w klasach pochodnych dziedziczących z klasy bazowej. Informacja taka jest umieszczana w specjalnej tablicy, której adres musi być przechowywany w każdym obiekcie klasy polimorficznej (posiadającej chociaż jedną metodę wirtualną). W związku z tym, taki obiekt musi być większy, co powoduje narzut pamięciowy.

Kiedy klasa posiada przynajmniej jedną ***czystą* funkcję wirtualną**, nazywana jest klasą abstrakcyjną. *Czysta* funkcja wirtualna staje się *czysta* jeśli jej inicjator ma postać = 0. **Metoda *czysto* wirtualna powinna być przesłonięta w klasie pochodnej**.

```
class KlasaAbstrakcyjna
{
public:
    virtual void metodaDoPrzesloniecia() = 0; // =0 czyli czysto wirtualna
};
```

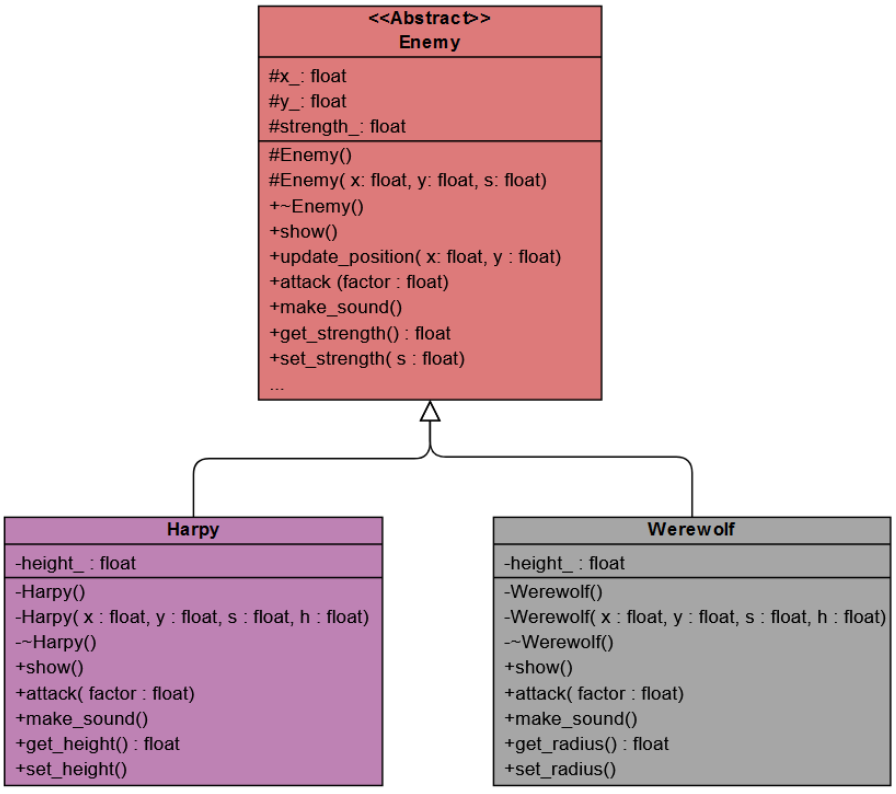
Jeżeli klasa pochodna nie przesłoni wszystkich metod czysto wirtualnych, również jest klasą abstrakcyjną. Nie można utworzyć obiektów klasy abstrakcyjnej. Może ona stanowić jedynie klasę bazową lub interfejs (szczególny rodzaj klasy abstrakcyjnej, zawierającej jedynie publiczne funkcje składowe, *czysto* wirtualne i wirtualny destruktor). **Kiedy metoda jest stworzona jako *czysto* wirtualna, nie jest ona w ogóle zdefiniowana. W przeciwieństwie do metody wirtualnej, której definicja musi wystąpić, chociażby miała być pusta.**

2 Struktura programu

Inspiracja: *„Ludzie (...) lubią wymyślać potwory i potworności. Sami sobie wydają się wtedy mniej potworni (...) Wtedy jakoś lżej im się robi na sercu. I łatwiej im żyć."* A.Sapkowski

Stwórz program, który będzie modelował różnych wrogów/potwory w grze fantasy. Niech posiada on trzy klasy: `Enemy` (wróg), `Harpy` (Harpia), `Werewolf` (Wilkołak). Klasy `Harpy`, `Werewolf` dziedziczą po klasie `Enemy` poprzez *public*. Na rysunku 1 przedstawiony został diagram obrazujący zależności między klasami, posiadane przez nie pola oraz metody.

Pisząc program pamiętaj o **podziale na pliki nagłówkowe**. Deklaracje klas umieść w pliku nagłówkowym (nadaj mu tę samą nazwę, co nazwa klasy), definicje metod umieszczaj w pliku źródłowym (również nadaj mu nazwę klasy).



Rysunek 1: Diagram klas, struktura programu

1. **Klasa bazowa** `Enemy` będzie posiadać:
   - pola *protected*: `x_`, `y_`, które reprezentują odpowiednio położenie na mapie oraz `strength_`, czyli siłę wroga, wszystkie są typu *float*,
   - Metody *protected*:
     - konstruktory: domyślny nadający wartości `0.0` oraz wieloargumentowy.
   - Metody *public*:
     - destruktor, wyświetlający tekst *Enemy destroyed!*,
     - metody *get* oraz *set* dla każdego pola klasy,
     - aktualizująca pozycję wroga (`x_`, `y_`) poprzez dodanie do współrzędnych odpowiednio wartości `x` i `y` przekazanych w argumentach funkcji,
     - (czysto wirtualna) wyświetlająca wszystkie parametry danego wroga,
     - (czysto wirtualna) reprezentująca atak, jako argument przyjmuje pewien współczynnik `float factor`,
     - (czysto wirtualna) reprezentująca dźwięk wydawany przez wroga podczas ataku,
2. **Klasa pochodna** `Harpy` posiada:
   - pole *private*: `height_`, które reprezentuje wysokość lotu.
   - Metody *public*:
     - konstruktor domyślny, bezparametrowy, nadający polu wartosc `0.0`,
     - konstruktor wieloargumentowy, który inicjalizuje pole wartościami przekazanymi w argumencie (przypomnienie z poprzednich zajęć: pamiętaj, że klasa dziedziczy inne pola, jak nadać im wartość?),
     - destruktor, który wyświetla: *Harpy destroyed*,
     - *getter* oraz *setter* dla pola klasy,
     - metodę wyświetlającą wszystkie parametry, przykładowo: *The Harpy is at point (2,2) and its height is 1*,
     - metodę odpowiadającą za wydanie dźwięku, która wyświetla na standardowym wyjściu napis: *Harpy's yell*,
     - metodę reprezentującą atak (atak Harpii powoduje zmniejszenie jej siły (odjęcie) wartości uzyskanej poprzez przemnożenie przez podany w argumencie współczynnik tej siły (jeżeli siła wynosi 100 punktów, a współczynnik wynosi 0.1, to 100 - 0.1*100 = 90) i jednoczesne obniżenie lotu, czyli ustawienie jej wysokości na 1.85 oraz wydanie dźwięku),
3. **Klasa pochodna** `Werewolf` posiada:
   - pola *private*: `radius_`, reprezentujące zasięg ręki Wilkołaka przy danym ruchu/ataku.
   - Metody *public* analogiczne, jak w klasie `Harpy`. Oczywiście zamiast np. wyświetlania wysokości teraz należy wyświetlić promień oraz dźwięk wilkołaka to: *Werewolf's growl!*,
   - Atak Wilkołaka polega również na pomniejszeniu siły oraz zwiększeniu zasięgu ręki (promienia) 1.5 raza, a także wydanie dźwięku.

   Uwagi:
   - Metody klas pochodnych (oczywiście poza konstruktorami, destruktorami, metodami `get` oraz `set`) nazwij tak samo, ma wystąpić przesłanianie się metod. Chcemy przetestować polimorfizm w praktyce.
   - Tworząc konstruktory korzystaj z listy inicjalizacyjnej. (przypomnienie z poprzednich zajęć)

3 Zadania

W zależności, jaką ocenę chcesz uzyskać, wykonaj podane polecenia dotyczące danej oceny.

Polecenia *ocena 3.0*

- Utwórz wskaźnik klasy bazowej i wstępnie zainicjuj go poprzez `nullptr`.
- Utwórz obiekty dynamiczne klas pochodnych stosując konstruktory wieloargumentowe.
- Następnie wskaźnikowi klasy bazowej przypisz wskaźnik klasy pochodnej `Harpy` i wykorzystaj go do wywołania funkcji: wyświetlającej, uaktualniającej położenie, reprezentującej atak.
- Powtórz to samo dla obiektu klasy `Werewolf`.
- Pamiętaj o zwolnieniu pamięci.

Odpowiedz w komentarzu:

1. Jak myślisz, dlaczego pola klasy `Enemy` są *protected*, a nie `private`?
2. Czy można wskaźnikowi klasy pochodnej przypisać wskaźnik klasy bazowej?

Polecenia *ocena 4.0*

- Stwórz dynamiczną tablicę czterech wskaźników typu `Enemy` (tablica wskaźników, w której każdy indeks tablicy wskazuje na blok pamięci, dynamiczna tablica wskaźników 2D).
- Stwórz dwa obiekty dynamiczne typu `Harpy` oraz dwa obiekty dynamiczne typu `Werewolf`. Użyj konstruktorów wieloargumentowych.
- Wskaźnikom z tablicy przypisz utworzone obiekty (do każdej komórki tablicy, w której znajduję się wskaźniki tpu `Enemy`, przypisz dany wskaźnik utworzonego dynamicznie obiektu).
- Wykorzystując pętle do przejścia po tablicy zaprezentuj działanie poszczególnych funkcji utworzonych w klasie `Harpy` oraz `Werewolf`. Sprawdź, czy działają poprawnie.
- Pamiętaj o zwolnieniu pamięci po utworzonych obiektach i tablicy.
- Jak myślisz, dlaczego pola klasy `Enemy` są *protected*, a nie `private`?

Polecenia *ocena 5.0*

- Wykonaj zadania na ocenę 4.0 i odpowiedz na poniższe pytania w komentarzu.
- Czym się różnią metody czysto wirtualne od wirtualnych?
- Które ze stworzonych klas są klasami abstrakcyjnymi? Uzasadnij odpowiedź.